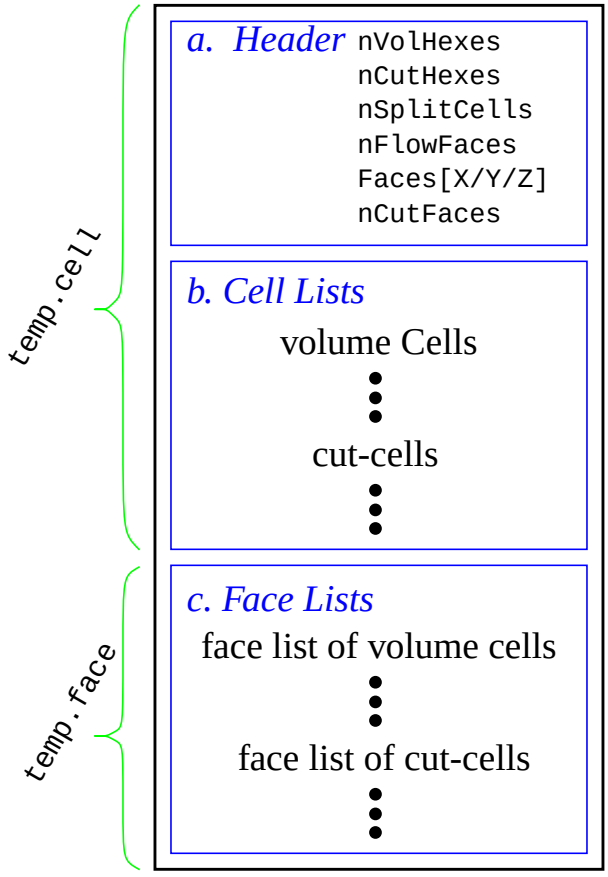
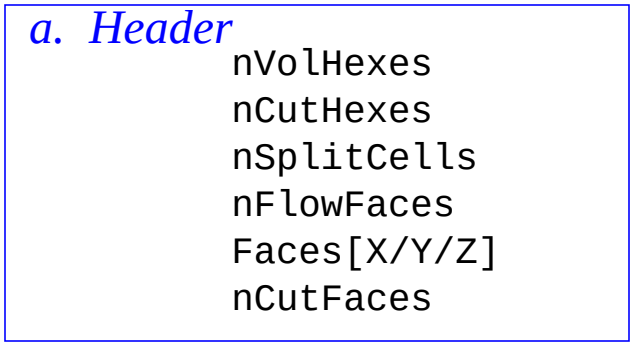
The Composition of Cart3D Output Files

default filename: `<mesh.c3d>`

Output files are written in *C* binary using the fwrite from <stdio.h>. As the "*cubes*" mesh generator executes, it writes two separate temporary files "`temp.cell`" and "`temp.face`" which are opened during volume mesh generation, and appended during cut-cell mesh generation. At the end of execution, *cubes* concatenates these two files together to make a single output file (default name is `mesh.c3d`). The first sketch shows the general structure of this file.





a. Header information:

`nVolHexes:` Number of fully "FLOW" Cartesian hexahedra in the volume mesh (not counting any "INTERSECTED" cells) (type "**`int`**")

`nCutHexes:` Number of Cartesian hexahedra which actually intersect the solid wall boundaries. (type "**`int`**")

`nSplitCells:` The number of polyhedra into which the "split-cells" are cut. (see figure). (type "**`int`**")

`nFlowFaces:` The number of flow-through, non-intersected Cartesian faces connecting the `nVolHexes` FLOW hexes in the volume mesh. Note that only faces whose adjacent cells have status FLOW | FLOW are counted by `nFlowFaces`. (type "**`int`**")

`Faces[X/Y/Z]:` The number of non-intersected hex faces in each Cartesian direction in the face list. The list of un-cut faces is sorted, so that retaining these three integers uniquely identifies which faces are in which direction. (type "**`int`**")

`nCutFaces:` The total number of flow-through faces associated with all the cut-cells. This is the number of face-polys of the cut-cells, and includes non-intersected faces whose adjacent cells have status "INTERSECTED | FLOW". (type "**`int`**")

b. Cell Lists:

Two arrays of structures describe cell-based information. (1) The first describes the Cartesian cell information only (location, name, size, etc.). It is organized with the list of `nVolHexes` volume cells (type "FLOW") before the list of "INTERSECTED" hexes which cut the boundary. (2) The second array contains additional information about the cut-cells. Since some of these may be split into many different control volumes (split-cells) there are a total of `nCutHexes` + `nSplitCells` entries in this array. The first nCutHexes entries correspond (1-to-1) with the last `nCutHexes` of the Cartesian cell list. The remainder of entries are split-Cells.

```
#define DIM 3                                   /*              ...for 3 space dimensions */
                                                /*          ---- declare low storage hexs ---- */
typedef struct TinyHexStructure ts_TinyHex;
typedef ts_TinyHex *p_tsTinyHex;
                                                /*            --  define tiny hex type flags -  */
typedef enum {UNSET_HEX, FLOW_HEX, CUT_HEX, SPLIT_HEX } tinyHexType;

struct TinyHexStructure {                       /*        .....define the complete structure      */
  INT64       name;                             /*          -->name tells position in mesh        */
```

```
  char        ref[DIM];                  /*          -->Num. of cell divisions in X,Y,Z   */
  byte        hexType;                   /*              --> (byte) cast of tinyHexType   */
};
```

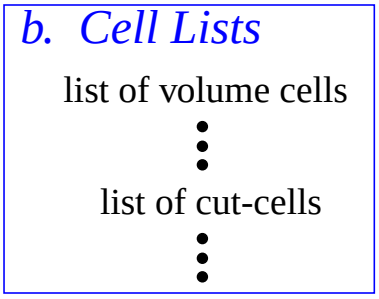
Cell Lists (continued):

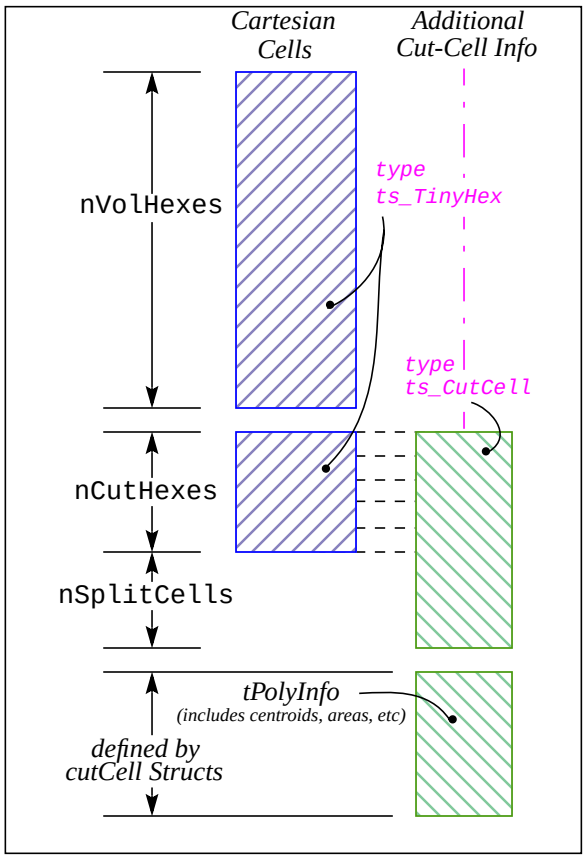
```
typedef double   dpoint3[DIM];      /*                  -- define double 3D point --------*/
typedef dpoint3 *p_dpoint3;
                                    /*      ---- declare annotated cut-cell with full surf info ----*/
typedef struct CutCellStruct ts_CutCell;
typedef ts_CutCell *p_tsCutCell;

struct ts_CutCell{                  /*      ...define the complete cut-cell structure              */
   int            nIntTri;          /*        -->  no. of tri's connected to cell                  */
   dpoint3        normal;           /*        -->  agglomerated surf norm vec.                     */
   dpoint3        centroid;         /*        -->  volume centroid of cell                         */
   double         volume;           /*        -->  volume  of cell                                 */
   int            splitIndex;       /*        -->  splitIndex < 0: hex not split                   */
                                    /*             0≤splitIndex < nCutHexes: pt to splitParent     */
                                    /*             splitIndex ≥ nCutHexes: pt to first splitKid    */
   int           *p_IntTriList;     /*        -->  ptr to list of indices of intersect triangles   */
   p_dpoint3      p_centroids;      /*        -->  ptr to list of centroids of tPolys              */
   double        *p_area;           /*        -->  ptr to list of areas of tPolys                  */
};
```

The "Cell Lists" are written to the file with 4 sequential writes.

1. Write the Cartesian hex info for the `nVolHexes` un-cut (volume mesh) cells.

```
  for(j=0;j<nVolHexes;j++){
```





```
    fread(&p_VolHexes[j].name,  sizeof(INT64),  1,p_InputMeshstrm);
    fread(&p_VolHexes[j].ref[0],sizeof(char), DIM,p_InputMeshstrm);
  }
```

2. Write the Cartesian hex info for the `nCutHexes` cut (body intersecting) cells.

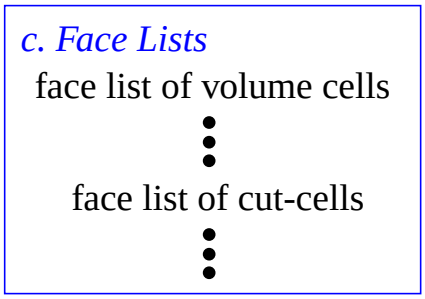
```
  for(j=0;j<nCutHexes;j++){
    fread(&p_CutHexes[j].name,  sizeof(INT64),  1,p_InputMeshstrm);
    fread(&p_CutHexes[j].ref[0],sizeof(char), DIM,p_InputMeshstrm);
  }
```

3. Begin filling out the cut-cell information.

```
{
  int         j, totalTri = 0;
  p_tsCutCell p_CutCells;

  for(j=0;j<nCutHexes+nSplitCells;j++){
    fwrite(&totalTri,                sizeof(int)    ,1,p_TmpCellFile);
    fwrite(&p_CutCells[j].nIntTri,   sizeof(int)    ,1,p_TmpCellFile);
    fwrite( p_CutCells[j].normal,    sizeof(dpoint3),1,p_TmpCellFile);
    fwrite( p_CutCells[j].centroid,  sizeof(dpoint3),1,p_TmpCellFile);
    fwrite( p_CutCells[j].volume,    sizeof(double ),1,p_TmpCellFile);
    fwrite(&p_CutCells[j].splitIndex,sizeof(int)    ,1,p_TmpCellFile);
    totalTri+=p_cCells[j].nIntTri;
  }
}
```

4. Write the triangle/tPoly info for each cut-Cell.

```
{
  int i,j;
  p_tsCutCell p_CutCells;

  for(j=0;j<nCutHexes+nSplitCells;j++){
   for(i=0;i<p_CutCells[j].nIntTri;i++){             /*       <-- No. of tris linked to cell */
     fwrite(&p_CutCells[j].p_IntTriList[i],sizeof(int)    ,1,p_TmpCellFile);
     fwrite( p_CutCells[j].p_centroids[i], sizeof(dpoint3),1,p_TmpCellFile);
     fwrite(&p_CutCells[j].p_area[i]    ,  sizeof(double) ,1,p_TmpCellFile);
    }
  }
}
```



c. Face Lists

Cell-to-cell mesh connectivity is stored through face-lists. For compactness, these come in two specialized datatypes, (1) Cartesian face lists - this is a minimal face structure, and is used only for non-intersected mesh faces, (FLOW | FLOW). It has only two entries - the indices of the un-cut Cartesian cells on either side. This list is sorted so that the first `Faces[X]` faces have normal vectors in the *x* direction, and these are followed by the *y* and *z* faces. (2) The second datatype is a more general Cartesian face structure which includes the face area, centroid, etc.. for cut cells. Indexing into the cell lists assumes a contiguous, sequential cell numbering, starting with the un-cut volume cells, through the split cells. Obviously, the first `nVolHexes` indices refer to volume cells, and higher indices refer to cut or split cells. Faces at topological mesh boundaries (far-field, symm etc..) have cells on only one side, degenerate cases are denoted with a `NO_CELL_FLAG_INDX = -1.`

```
                                                /*          ---- declare low storage Cart Face ---- */
typedef struct VolumeFaceStructure tsVface;
typedef tsVface *p_tsVface;

struct VolumeFaceStructure {                    /*              ...define the complete structure  */
  int     adjCell[2];                           /*            --> 0 = Index of cell on low  side  */
};                                              /*                1 = Index of cell on high side */

typedef struct CutFaceStructure tsCface;        /*     ---- declare Cut Cart Face  (lots of info) ----*/
typedef tsCface *p_tsCface

struct CutFaceStructure {                       /*              ...define the complete structure  */
  int       adjCell[2];                         /*            --> 0 = low, 1 = high               */
  dpoint3   centroid;                           /*            --> area centroid of face           */
  double    area;                               /*            -->  area of face                   */
  char      dir;                                /*            --> 0 = X, 1 = Y, 2 = Z             */
};
```

Both of these lists are written with a single write statement.

```
{
  p_tsVface p_Faces;
  p_tsCface p_cFaces;
                                  /* ...1. Write out Cartesian faces for volume mesh (uncut faces)      */
    fwrite( p_Faces,  sizeof(tsVface), nFlowFaces,p_TmpFaceFile);
                                  /* ...2. Write out Cartesian face information for cut Cells           */
    for(j=0;j<nCutFaces;j++){     /*              (just like we wrote 'em)                             */
      fwrite( p_cFaces[j].adjCell, sizeof(int    ), 2, p_InputMeshstrm);
      fwrite( p_cFaces[j].centroid,sizeof(dpoint3), 1, p_InputMeshstrm);
      fwrite(&p_cFaces[j].area    ,sizeof(double ), 1, p_InputMeshstrm);
      fwrite(&p_cFaces[j].dir     ,sizeof(char   ), 1, p_InputMeshstrm);
     }
}
```